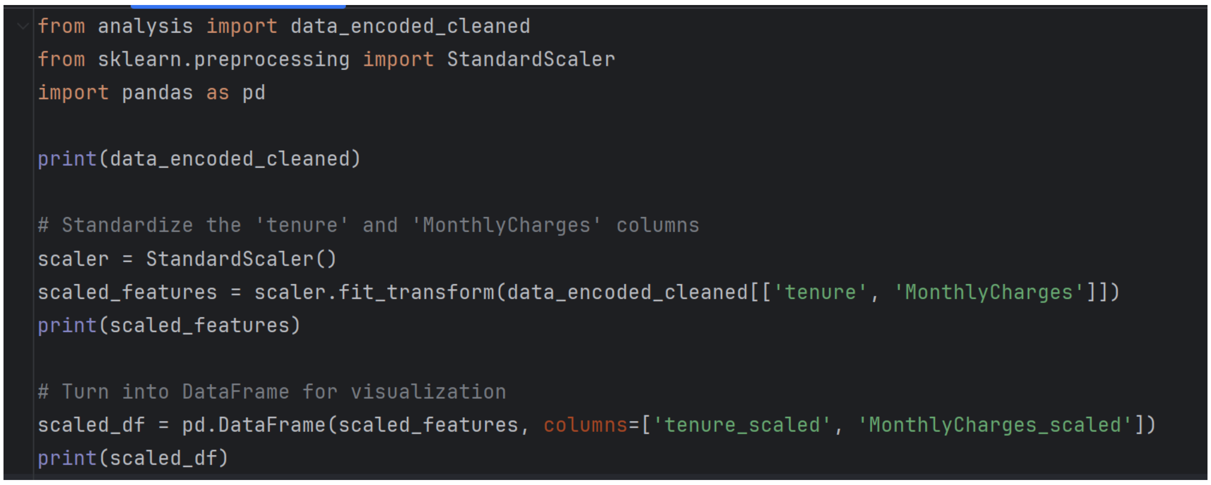
Optimal Number of Cluster

Our primary responsibility is to employ advanced clustering algorithms to segment customers based on similarities in tenure and monthly charges. Therefore, we are clustering based on two columns: “tenure” and “MonthlyCharges”.

Before applying clustering algorithms, we need to prepare the data:

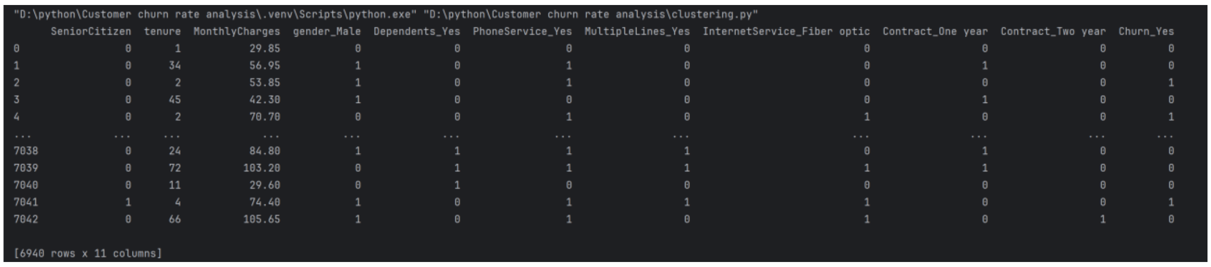
- Data cleaning
- Data standardization

The dataset has been cleaned by our team’s Data Engineer. Now, we only need to standardize (scale) two columns from the dataset: “**tenure**” and “**MonthlyCharges**”.



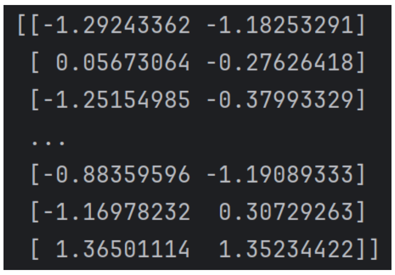
```python
from analysis import data_encoded_cleaned
from sklearn.preprocessing import StandardScaler
import pandas as pd

print(data_encoded_cleaned)

# Standardize the 'tenure' and 'MonthlyCharges' columns
scaler = StandardScaler()
scaled_features = scaler.fit_transform(data_encoded_cleaned[['tenure', 'MonthlyCharges']])
print(scaled_features)

# Turn into DataFrame for visualization
scaled_df = pd.DataFrame(scaled_features, columns=['tenure_scaled', 'MonthlyCharges_scaled'])
print(scaled_df)
```

Output:



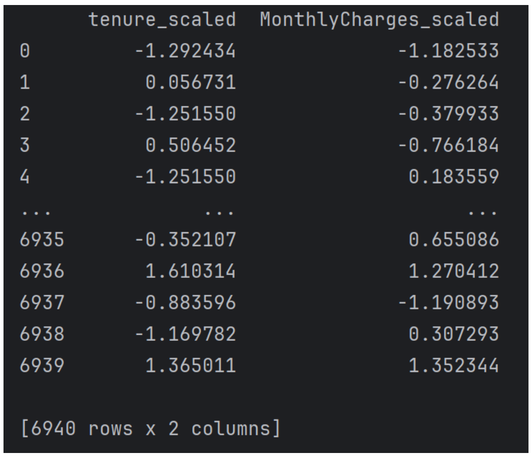
```
"D:\python\Customer churn rate analysis\.venv\Scripts\python.exe" "D:\python\Customer churn rate analysis\clustering.py"
      SeniorCitizen  tenure  MonthlyCharges  gender_Male  Dependents_Yes  PhoneService_Yes  MultipleLines_Yes  InternetService_Fiber optic  Contract_One year  Contract_Two year  Churn_Yes
0                 0       1           29.85            0               0                 0                  0                            0                  0                  0          0
1                 0      34           56.95            1               0                 1                  0                            0                  1                  0          0
2                 0       2           53.85            1               0                 1                  0                            0                  0                  0          1
3                 0      45           42.30            1               0                 0                  0                            0                  1                  0          0
4                 0       2           70.70            0               0                 1                  0                            1                  0                  0          1
...             ...     ...             ...          ...             ...               ...                ...                          ...                ...                ...        ...
7038              0      24           84.80            1               1                 1                  1                            0                  1                  0          0
7039              0      72          103.20            0               1                 1                  1                            1                  1                  0          0
7040              0      11           29.60            0               1                 0                  0                            0                  0                  0          0
7041              1       4           74.40            1               0                 1                  1                            1                  0                  0          1
7042              0      66          105.65            1               0                 1                  0                            1                  0                  1          0

[6940 rows x 11 columns]
```

Show the cleaned dataset before scaling



```
[[-1.29243362 -1.18253291]
 [ 0.05673064 -0.27626418]
 [-1.25154985 -0.37993329]
 ...
 [-0.88359596 -1.19089333]
 [-1.16978232  0.30729263]
 [ 1.36501114  1.35234422]]
```

A NumPy array containing scaling result


```
      tenure_scaled  MonthlyCharges_scaled
0         -1.292434              -1.182533
1          0.056731              -0.276264
2         -1.251550              -0.379933
3          0.506452              -0.766184
4         -1.251550               0.183559
...             ...                    ...
6935      -0.352107               0.655086
6936       1.610314               1.270412
6937      -0.883596              -1.190893
6938      -1.169782               0.307293
6939       1.365011               1.352344

[6940 rows x 2 columns]
```

Turn the NumPy array into DataFrame for better visualization

Determine optimal number of clusters using the elbow method

The Elbow Method is used to determine the optimal number of clusters K in K-Means clustering. It helps find the value of K where adding more clusters does not significantly improve the quality of the clustering. This is done by plotting the WCSS (Within-Cluster Sum of Squares) for different values of K and looking for an "elbow" point in the plot.

Steps for the Elbow Method:

1. Run the K-Means algorithm with different values of K (e.g., 1 to 10).
2. Calculate the WCSS for each value of K. WCSS measures the compactness of the clusters.
3. Plot the WCSS against the number of clusters (K):
   - The x-axis represents the number of clusters (K).
   - The y-axis represents the WCSS.
4. Identify the "elbow point":
   - The point where the rate of decrease in WCSS slows down significantly.
   - This indicates that increasing the number of clusters beyond this point does not significantly reduce the WCSS.

Code:

```python
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Use the Elbow Method to find the optimal number of clusters
wcss = []
for k in range(1, 11):  # Trying values of K from 1 to 10
    # kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=300, n_init=10, random_state=42)
    kmeans.fit(scaled_features)
    wcss.append(kmeans.inertia_)

# Print the result of wcss array for visualization
print(wcss)

# Plot the WCSS for each k
plt.plot( *args: range(1, 11), wcss, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.title('Elbow Method for Optimal K')
plt.show()
```

Code explanation:

```python
kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=300, n_init=10, random_state=42)
```

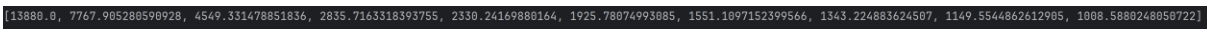
- creates an instance of the KMeans clustering algorithm from the sklearn.cluster module in Python
- Parameters explanation:
    - **n_clusters**: number of clusters to form (K)
        - No default value
    - **init**: method for initializing the centroids ('k-means++', 'random', etc.)
        - Default value: ‘k-means++’
    - **max_iter**: Maximum number of iterations in a single run
        - Default value: 300
    - **n_init**: number of times the algorithm will run with different initial centroids
        - Default value: 10
    - **random_state**: seed for the random number generator to ensure reproducibility
        - No default value

```python
kmeans.fit(scaled_features)
```

- KMeans performs the clustering process and the model is fitted to the data.
- We have the following key results stored in the K-Means model after running kmeans.fit():
    - **kmeans.inertia_**
        - Represents WCSS
        - We can compare the inertia values for different numbers of clusters (K) using the Elbow Method to find the optimal number of clusters.
    - kmeans.cluster_centers_

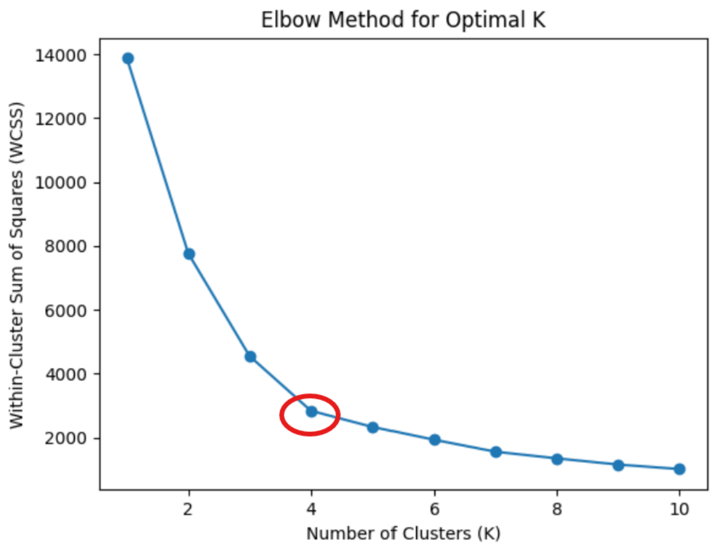
        - This attribute contains the coordinates of the centroids for each cluster.
    - kmeans.labels_
        - This attribute contains the cluster assignment for each data point in the dataset
    - kmeans.n_iter_
        - This attribute shows the number of iterations the algorithm ran before reaching convergence

Output:



```
[13880.0, 7767.905280590928, 4549.331478851836, 2835.7163318393755, 2330.24169880164, 1925.78074993085, 1551.1097152399566, 1343.224883624507, 1149.5544862612905, 1008.5880248050722]
```

wcss array



Plot wcss against different values of K (number of cluster)

Based on the graph above, we can see that starting from K=4, WCSS decreases not as much as before. This determines the **optimal number of cluster is 4**.

Why choose the "elbow" as the optimal number of clusters?

- Before the "elbow," adding more clusters significantly reduces the WCSS, which means that increasing the number of clusters improves the compactness of the clusters.
- After the "elbow," the improvement in WCSS diminishes, indicating diminishing returns in terms of clustering quality. Adding more clusters beyond this point only marginally reduces the WCSS.
- At the "elbow," the balance between the number of clusters and the WCSS is optimal.
- Increasing K beyond the elbow point results in more complex models with more clusters but does not significantly improve the clustering quality.
- Choosing too many clusters (K) may lead to overfitting, where the clusters are too specific and do not generalize well to new data.